HANDLING INPUT MULTIPLEXATING CONTROLLERS

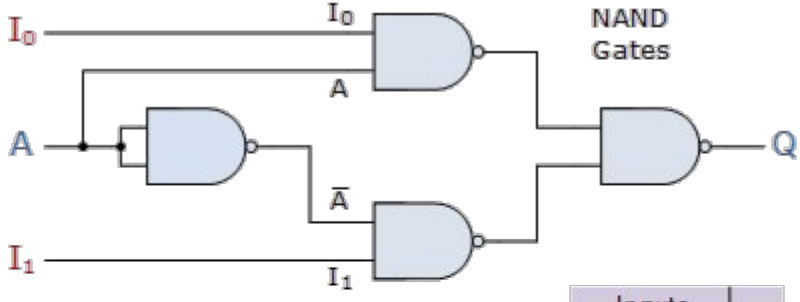
Julio Sánchez Cuenca

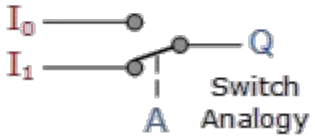
Index

1. Multiplexing
2. InputProcessor
3. InputMultiPlexer
4. How does InputProcessor work?
5. How does InputMultiPlexer work?
6. What Purpose does it have?

MULTIPLEXING

Multiplexing is the generic term used to describe the operation of sending one or more analogue or digital signals over a common transmission line at different times or speeds.





Truth Table

Inputs			
A	I_1	I_0	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

INPUTPROCESSOR

An InputProcessor is used to receive input events from the keyboard and the touch screen (mouse on the desktop).

INPUTMULTIPLEXER

The InputMultiplexer will hand any new events to the first InputProcessor that was added to it. If that processor returns false from the method invoked to handle the event, this indicates the event was not handled and the multiplexer will hand the event to the next processor in the chain.

BUT HOW DOES INPUTPROCESSOR WORK?

You must create a class that implements InputProcessor, which will implement these methods that control the entries of different kinds.

Modifier and Type	Method and Description
boolean	**keyDown**(int keycode) Called when a key was pressed
boolean	**keyTyped**(char character) Called when a key was typed
boolean	**keyUp**(int keycode) Called when a key was released
boolean	**mouseMoved**(int screenX, int screenY) Called when the mouse was moved without any buttons being pressed.
boolean	**scrolled**(int amount) Called when the mouse wheel was scrolled.
boolean	**touchDown**(int screenX, int screenY, int pointer, int button) Called when the screen was touched or a mouse button was pressed.
boolean	**touchDragged**(int screenX, int screenY, int pointer) Called when a finger or the mouse was dragged.
boolean	**touchUp**(int screenX, int screenY, int pointer, int button) Called when a finger was lifted or a mouse button was released.

```
11  public class EntradaCocheRaton extends InputAdapter {
12
13          private ControladorVirtual controlador;
14
15          public EntradaCocheRaton(ControladorVirtual controlador) {
16                  this.controlador = controlador;
17          }
18
19          /** touchUp = al dejar de pulsar. */
20          @Override
21          public boolean touchUp(int screenX, int screenY, int pointer, int button) {
22                  controlador.obedeceRaton = true;
23                  controlador.objetivoX = screenX;
24                  return true;
25          }
26
27  }
```

BUT HOW DOES INPUTMULTIPLEXER WORK?

In the class that implements ApplicationListener, we will create the InputMultiplexer and add the InputProcessor that we want, as if it were a list. The InputMultiplexer will go through the InputProcessor until some event returns true, then it will stop its search.

```
InputMultiplexer multiplexer = new InputMultiplexer();
multiplexer.addProcessor(new MyUiInputProcessor());
multiplexer.addProcessor(new MyGameInputProcessor());
Gdx.input.setInputProcessor(multiplexer);
```

WHAT PURPOSE DOES IT HAVE?

It can be used, for example, in first-person shooter games, to separate the use of the keyboard and the mouse in different InputProcessors. So to be able to handle the movement of the character, as well as move the camera of it.

```
23          @Override
24          public void show() {
25                  coche = new Coche();
26                  coche.setPosition(50, 50);
27
28                  // Creo los controladores.
29                  controlador = new ControladorVirtual();
30                  teclado = new EntradaCocheTeclado(controlador);
31                  raton = new EntradaCocheRaton(controlador);
32
33                  // Para poder usar ambos a la vez creo multiplexores.
34                  multiplexor = new InputMultiplexer();
35                  multiplexor.addProcessor(raton);
36                  multiplexor.addProcessor(teclado);
37
38                  // setInputProcessor = poner un controlador o un multiplexor.
39                  Gdx.input.setInputProcessor(multiplexor);
40          }
41
```